Homework

Basic:

1. 用户能通过左键点击添加Bezier曲线的控制点，右键点击则对当前添加的最后一个控制点进行消除
2. 工具根据鼠标绘制的控制点**实时**更新Bezier曲线。

Hint: 大家可查询捕捉mouse移动和点击的函数方法

Bonus:

1. 可以动态地呈现Bezier曲线的生成过程。

1. 贝塞尔曲线

首先参考课程 PPT 上对于 Bezier curve 的定义，根据输入的 n 个控制点可以得到曲线的方程 Q(t)，其中 t 的范围为 0-1 闭区间。

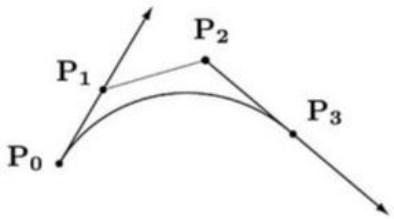
The definition of Bézier curve

- Bézier curve本质上是由**调和函数（Harmonic functions）**根据**控制点（Control points）**插值生成。其参数方程如下：

$$Q(t)=\sum_{i=0}^{n}P_iB_{i,n}(t),\quad t\in[0,1]$$

- 上式为n次多项式，具有$n+1$项。其中，$P_i(i=0,1\ldots n)$表示特征多边形的$n+1$个顶点向量；$B_{i,n}(t)$为**伯恩斯坦（Bernstein）基函数**，其多项式表示为：

$$B_{i,n}(t)=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i},\quad i=0,\ 1\ldots n$$



为了实现曲线绘制的动态过程，我们还需要使用曲线的递归性：

- Recursive(递归性)：

$$B_{i,n}(t)=(1-t)B_{i,n-1}(t)+tB_{i-1,n-1}(t),\quad (i=0,1,\ldots,n)$$

根据以上数学表达，我们定义用于绘制 Bezier 曲线的类 Bezier.cpp
对应头文件 Bezier.hpp 中的定义如下：

私用成员变量 vertices 是以 glm::vec3 为元素的 vector 变量，存储的是曲线中所有的控制点；

push 方法用来添加控制点；pop 方法减少一个最后添加的控制点。
linearCombination 方法接受一组点的集合，返回经这些点亮两线性组合后的到的点，t 变量用来指定线性组合中的比重大小。若输出点的规模为 n，返回 n-1 个点。
bezierCurve 用来绘制 n 个控制点对应的贝塞尔曲线，n 用来指定返回的贝塞尔曲线上点的个数。最终返回的点的数量为 n+1。
getVertices 用来获得当前所有的控制点。

私有的成员函数中的 C 和 B 函数对应上一页贝塞尔曲线定义中的组合数与 B 函数。

```
#ifndef BEZIER_HPP
#define BEZIER_HPP

#include <vector>
#include <cmath>
#include <glm/glm.hpp>
#include <iostream>

class Bezier {
public:
    Bezier();
    bool push(glm::vec3 point);
    bool pop();
    std::vector<glm::vec3>
    linearCombination(std::vector<glm::vec3> vertices, float t);
    std::vector<glm::vec3> bezierCurve(int n);
    std::vector<glm::vec3> getVertices();
private:
    std::vector<glm::vec3> vertices;
    int C(int n, int i);
    float B(int n, int i, float t);

};

#endif
```

Bezier.cpp 具体实现如下：

这里实现的细节中，push 方法接受一个控制点，首先和最后一个控制点比较是否相同，如果相同就拒绝将新控制点加入 vertices 中，这么做是为了避免重复的控制点加入。但是这里并没有将新加入的控制点与 vector 中所有的点进行比较。实际中不排除此情况的发生。

pop 时若 vertices 非空直接移除最后一个元素即可。

这里在计算组合数 C(n,i)时，没有计算阶乘，而是使用递归，利用公式：C(n,i) = C(n-1,i-1) + C(n-1, i)计算。但是实际中当控制点数目过大时还是难以快速计算出组合数。所以这里一个优化的方向是采用缓存，这次没有实现。

```
#include "Bezier.hpp"

Bezier::Bezier()
{
}

bool Bezier::push(glm::vec3 point)
{
    int size = vertices.size();
    if (size == 0) {
        vertices.push_back(point);
        return true;
    }

    if (vertices[size - 1] == point) {
        return false;
    }
    vertices.push_back(point);
    return true;
}

bool Bezier::pop()
{
    if (vertices.size() == 0) {
        return false;
    }

    vertices.pop_back();
    return true;
}

std::vector<glm::vec3>
Bezier::linearCombination(std::vector<glm::vec3> points, float t)
{
    std::vector<glm::vec3> res;
```

```
    for (int i = 0; i < points.size() - 1; ++i) {
        res.push_back(points[i] * t + points[i + 1] * (1 - t));
    }
    return res;
}

std::vector<glm::vec3> Bezier::bezierCurve(int num)
{
    int n = vertices.size() - 1;
    std::vector<glm::vec3> res;
    if (n < 2) {
        return res;
    }

    for (int i = 0; i <= num; ++i) {
        float t = (float)i / num;
        glm::vec3 p(0.0, 0.0, 0.0);
        for (int j = 0; j <= n; ++j) {
            p += vertices[j] * B(n, j, t);
        }
        res.push_back(p);
    }
    return res;
}

std::vector<glm::vec3> Bezier::getVertices()
{
    return vertices;
}

int Bezier::C(int n, int i)
{
    if (i > n)
        return 0;
    else if (i == n || i == 0)
        return 1;
    else
        return C(n - 1, i) + C(n - 1, i - 1);
}

float Bezier::B(int n, int i, float t) {
    return C(n, i) * pow(t, i) * pow(1 - t, n - i);
}
```

2. Opengl 实现绘制

顶点着色器并没有什么特殊的功能，将输入的顶点位置输出即可：

Shader.v

```
#version 330 core
layout (location = 0) in vec3 pos;

void main()
{
   gl_Position = vec4(pos.x, pos.y, pos.z, 1.0);
}
```

片段着色器中指定颜色：

Shader.f

```
#version 330 core
out vec4 color;

void main()
{
   color = vec4(1.0f, 1.0f, 1.0f, 1.0f);
}
```

为了使得产生的中间曲线和最终的贝塞尔曲线颜色有差别，绘制贝塞尔曲线时的颜色有所不同，使用绿色。

ShaderBezier.f

```
#version 330 core
out vec4 color;

void main()
{
   color = vec4(0.0f, 1.0f, 0.0f, 1.0f);
}
```

由于要获取鼠标点击的位置，我们需要变量存储鼠标的 x,y 坐标；
鼠标的相对于显示屏幕左上角的坐标分别为 lastX 和 lastY

```
const int SCREEN_WIDTH = 1024;
const int SCREEN_HEIGHT = 1024;

float lastX = SCREEN_WIDTH / 2.0f;
float lastY = SCREEN_WIDTH / 2.0f;
```

对于鼠标移动事件我们定义函数 mouseCallback 进行监听；每当鼠标位置移动，就将鼠标位置记录在相应变量中。

```
bool firstMouse = true;
void mouseCallback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    lastX = xpos;
    lastY = ypos;
}
```

同样对于鼠标点击事件我们定义函数 mouseButtonCallback 进行监听，若点击了鼠标左键，就将对应点 push 进 bezier 实例中，其中 bezier 实例是类 Bezier 的实例。而鼠标点击右键时，就调用 pop 函数将最后加入的控制点弹出。

对于点击鼠标中键，实现动态的显示贝塞尔曲线的生成过程，下面第 3 部分再做阐述。

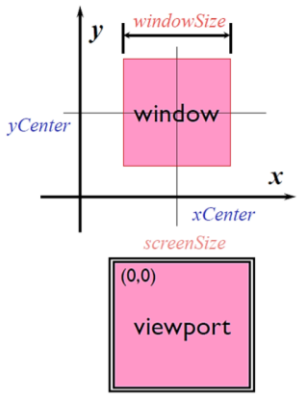
```
Bezier bezier;
void mouseButtonCallback(GLFWwindow* window, int button, int
action, int mods)
{
    if (action == GLFW_PRESS) switch (button)
    {
    case GLFW_MOUSE_BUTTON_LEFT:
        std::cout << "left" << std::endl;
        bezier.push(glm::vec3(lastX, lastY, 0));
        break;
    case GLFW_MOUSE_BUTTON_MIDDLE:
        {
            std::cout << "middle" << std::endl;
            if (state == PAUSE && lastState == PAUSE) {
                state = lastState = INCREASE;
            }
            else if (state == PAUSE && lastState != PAUSE) {
                state = lastState;
            }
            else if (state != PAUSE) {
                state = PAUSE;
            }
```

```
        }
        break;
    case GLFW_MOUSE_BUTTON_RIGHT:
        std::cout << "right" << std::endl;
        bezier.pop();
        break;
    default:
        return;
    }
    return;
}
```

定义函数 drawBezier 对贝塞尔曲线的中间曲线进行绘制；这里 n 个控制点就需要绘制 n 次，每次绘制的曲线是在上一次绘制曲线的基础上进行相邻点逐个线性组合得到的。

drawBezier 在循环中每次调用函数 drawLine，每次接受一组点，将这些点逐点连接成线。

这里需要注意的问题是在绘制点的过程中需要将相对于屏幕左上角的坐标转化为以屏幕中心为原点的坐标，并且 y 轴的方向也要进行反转。即将下图中下方的坐标系统转化为上方的坐标系统。具体方法代码中有体现。




```
float t = 0.25;
void drawBezier(Bezier &bezier, float t) {
    std::vector<glm::vec3> vertices = bezier.getVertices();
    int size = vertices.size();
    drawLine(vertices);
    for (int i = 0; i < size - 1; ++i) {
        std::vector<glm::vec3>
temp(bezier.linearCombination(vertices, t));
        vertices.swap(temp);
        drawLine(vertices);
    }
}
```

```
void drawLine(const std::vector<glm::vec3> &points) {
    if (points.size() == 0) {
        return;
    }

    std::vector<float> vertices;
    int w = display_w / 2;
    int h = display_h / 2;

    for (int i = 0; i < points.size(); ++i) {
        vertices.push_back((points[i].x - w)/ w);
        vertices.push_back((h - points[i].y)/ h);
        vertices.push_back(points[i].z);
    }

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float),
    &vertices[0], GL_STATIC_DRAW);
     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 *
    sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glBindVertexArray(VAO);
    if (points.size() > 1) {
        glDrawArrays(GL_LINE_STRIP, 0, points.size());
    }
    else {
        glDrawArrays(GL_POINT, 0, points.size());
    }
    glBindVertexArray(0);
}
```

main 函数

main.cpp

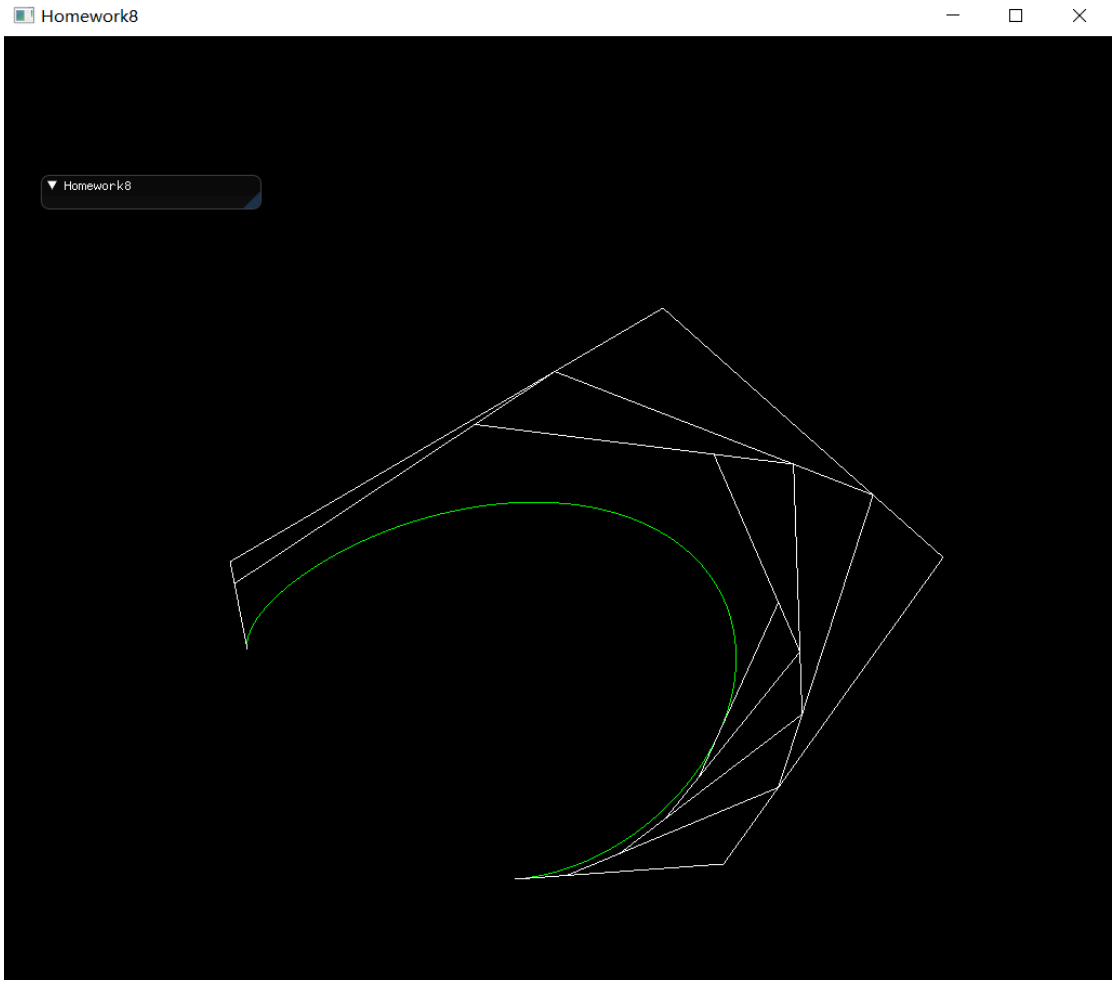
主函数在绑定监听函数后，在渲染循环中只需要调用 `drawBezier(bezier, t)` 和 `drawLine(bezier.bezierCurve(100))` 就可以绘制出曲线和中间的曲线。这里的 t 取 0.25。

```
unsigned int VAO;
unsigned int VBO;
float t = 0.25;

int main(int, char**)
{
```

```
    ……

    glfwSetCursorPosCallback(window, mouseCallback);
    glfwSetMouseButtonCallback(window, mouseButtonCallback);

    ……

    Shader myShader("Shader.v", "Shader.f");
    Shader myShaderBezier("Shader.v", "ShaderBezier.f");
    glEnable(GL_DEPTH_TEST);

    ……

    glfwMakeContextCurrent(window);
    glfwGetFramebufferSize(window, &display_w, &display_h);

    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glBindVertexArray(VAO);

    // Main loop
    while (!glfwWindowShouldClose(window))
    {
        ……

        glfwGetFramebufferSize(window, &display_w, &display_h);
        glViewport(0, 0, display_w, display_h);

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        myShader.useProgram();
        drawBezier(bezier, t);
        myShaderBezier.useProgram();
        drawLine(bezier.bezierCurve(100));
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
        glfwMakeContextCurrent(window);
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```

效果如下：

6 个控制点



3. 曲线动态变化

这里需要实现的功能是：

a. 点击鼠标中键可以开始或停止曲线动态变化

b. 当曲线参数 t 增大到最大后，自动停止变化，再次点击中键时 t 减小，曲线动态变化的方向相反。同样，t 减小到 0 时自动停止变化，再次点击中键 t 再开始增大。

下方的深红色代码是为了实现曲线动态变化添加的。

这里使用的基本思想就是有限状态机。定义不同的状态，并且鼠标中建点击和 t 变化至极值就是状态改变的条件。

为了在渲染循环中判断当前曲线是否在动态变化，需要为曲线定义一个状态，这定义枚举型变量 STATE，有三个不同的值，分别为 PAUSE,INCREASE 和 DECREASE；分别代表曲线变化的状态是停止，t 增大，t 减小。state 变量保存曲线当前状态，lastState 用于保存暂停之前曲线的状态，以便再次变化时从上次的状态中恢复。比如在曲线变化时点击了中建，此时曲线停止变化；再次点击中建时需要知道再次动态变化 t 变化的方向到底是增大还是减小的。

接下来就是在点击中键时改变状态的值，根据当前状态是否为 PAUSE 可以有不同

的动作；若当前状态为 PAUSE 还要继续判断 lastState 的状态来进行恢复。

初始使设置 state 与 lastState 的值均为 PAUSE。当 t 增大时，若 t 增大至 1 大于 1.0，则将 t 的值设置为 1.0，将 state 设置为 PAUSE 并将 lastState 赋值为 DECREASE；那么再次点击就可以实现 t 减小。

```
enum STATE{PAUSE, INCREASE, DECREASE};
STATE state = PAUSE;
STATE lastState = PAUSE;

while (!glfwWindowShouldClose(window))
{
    glfwGetFramebufferSize(window, &display_w, &display_h);
    glViewport(0, 0, display_w, display_h);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    float delta = 0.005;
    if (state == INCREASE) {
        t += delta;
        if (t > 1) {
            t = 1;
            state = PAUSE;
            lastState = DECREASE;
        }
    }
    if (state == DECREASE) {
        t -= delta;
        if (t < 0) {
            t = 0;
            state = PAUSE;
            lastState = INCREASE;
        }
    }

    myShader.useProgram();
    drawBezier(bezier, t);
    myShaderBezier.useProgram();
    drawLine(bezier.bezierCurve(100));
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
    glfwMakeContextCurrent(window);
    glfwSwapBuffers(window);
    glfwPollEvents();
}

void mouseButtonCallback(GLFWwindow* window, int button, int
action, int mods)
```

```
{
    if (action == GLFW_PRESS) switch (button)
    {
    case GLFW_MOUSE_BUTTON_LEFT:
        std::cout << "left" << std::endl;
        bezier.push(glm::vec3(lastX, lastY, 0));
        break;
    case GLFW_MOUSE_BUTTON_MIDDLE:
        {
            std::cout << "middle" << std::endl;
            if (state == PAUSE && lastState == PAUSE) {
                state = lastState = INCREASE;
            }
           else if (state == PAUSE && lastState != PAUSE) {
                state = lastState;
            }
            else if (state != PAUSE) {
                state = PAUSE;
            }
        }
       break;
    case GLFW_MOUSE_BUTTON_RIGHT:
      std::cout << "right" << std::endl;
       bezier.pop();
       break;
    default:
       return;
   }
   return;
}
```

实现的效果请看实验的视频。